Makalah Seminar Tugas Akhir

Implementasi dan Analisis Sistem Operasi Waktu-Nyata CHIBI Pada Mikrokontroler ARM

Eliyah Acantha Manapa[1], Ir. Endro Ariyanto, MT[2], Erwid Mustofa Jadied, ST, MT[3]
Jurusan Teknik Infromatika, Fakultas Teknik Informatika, Universitas Telkom
Jl. Telekomunikasi No. 1, Terusan Buah Batu, Bandung

Abstrak

Penggunaan *embedded system* dalam berbagai aspek kehidupan sudah banyak dilakukan, khususnya di bidang pengontrolan dan otomatisasi. *Embedded system* biasanya digunakan sebagai komponen inti dari sebuah produk untuk melakukan tugas komputasi yang *real time*. RTOS (*Real Time Operating System*) sendiri merupakan hasil pengembangan pada bidang IT sebagai sebuah sistem operasi yang diperuntukkan untuk operasi *real time*. Tujuan percobaan ini adalah untuk memperkenalkan prinsip kerja RTOS yang ditanam pada mikrokontroler ARM dengan tipe CORTEX-4 jenis STM32F4 *Discovery*. Dalam penelitian ini digunakan CHIBI RTOS yang *open source* sebagai sistem operasi waktu-nyata dan ARM sebagai mikrokontroler. Penelitian ini menggunakan metode eksperimen penanaman CHIBI RTOS pada Mikrokontroler ARM yang dilengkapi seperti komunikasi serial, LCD, LED dan *Push Button* dalam menjalankan Motor DC sebagai katalis dalam melihat *Preemptive Kernel*, *Semaphore* dan *Clock Tick* pada *Kernel*, kemudian diamati juga kecepatan dan waktu dari RTOS yang digunakan. Dari hasil pengujian diketahui bahwa CHIBI RTOS bisa ditanamkan pada mikrokontroler jenis ARM Cortex-M4. Pengujian pada *Preemptive Kernel*, *Semaphore* dan *Clock Tick* pada *Kernel* berfungsi dengan. Kedepannya akan lebih tepat melakukan pengujian yang lebih kompleks dan skenario lebih banyak.

Kata Kunci: *RTOS, task, scheduling, kernel, semaphore, clock tick, microcontroller ARM*

Abstract

The embedded systems have been used in various aspects of life, especially in the field of control and automation. Embedded systems are usually used as a core component of a product to perform computing tasks in real time. RTOS (Real Time Operating System) is a result of the development in the field of IT as an operating system intended for real time operation. The purpose of this experiment is to introduce the working principle of RTOS planted on ARM microcontroller of CORTEX-4 type STM32F4 Discovery. In this study the open source RTOS CHIBI was usesd as real time operating system and ARM as a microcontroller. Experimental method was used in the form of planting the CHIBI RTOS on ARM Microcontroller equipped with serial communication, LCD, LED and Push Button in running DC motors as a catalyst in view Preemptive Kernel, semaphores and Clock Tick on Kernel. The result shows that the CHIBI RTOS can be embedded in the microcontroller types of ARM Cortex-M4. Tests on Preemptive Kernel, Semaphores and Clock Tick on Kernel perform well. In the future, more will precise testings for more complex and various scenarios will be needed.

Keywords: *RTOS, task, scheduling, kernel, semaphore, clock tick, microcontroller ARM*

I. PENDAHULUAN

Perkembangan teknologi mikrokontroler dalam sistem kontrol yang melibatkan perangkat lunak bukan merupakan hal baru. Penanaman sistem mikrokontroler dalam peralatan sering disebut dengan *embedded system* (sistem tertanam). Penerapan sistem mikrokontroler dilakukan pada berbagai bidang seperti telekomunikasi, otomotif, peralatan rumah tangga, *robotic*, peralatan medis, maupun komputer. Sistem ini menuntut kinerja proses kendali yang semakin baik.

Pada perancangan sistem tertanam tersebut, ikut dimasukkan sebuah perangkat lunak sistem operasi yang dikenal sebagai *real time operating sistem* (RTOS). Sistem tertanam biasanya digunakan sebagai komponen inti dari sebuah produk dan dirancang untuk tujuan khusus guna melakukan satu atau banyak tugas dalam komputasi yang *real time*. Berdasarkan hal tersebut dibutuhkan bantuan perangkat lunak sistem operasi untuk mengkoordinasikan aplikasi sistem tertanam agar sistem tersebut bukan hanya dapat berjalan dengan baik tetapi juga bisa sesuai dengan waktu yang diinginkan, deterministik, *reliable* dan efisien.

RTOS sendiri merupakan hasil pengembangan pada bidang IT sebagai sebuah sistem operasi yang diperuntukkan untuk operasi *real time* yang kemudian bisa diadaptasikan untuk bidang otomatisasi dan pengontrolan. Pada sistem *real time*, sebuah *task* (proses) dapat diselesaikan dalam waktu tertentu yang bisa ditentukan oleh *user* sendiri pada saat proses perancangan sistem tersebut. RTOS berhubungan dengan *schedulling* (penjadwalan) yang memungkinkan pengerjaan beberapa *task* secara teratur sehingga kemungkinan bertabrakannya beberapa *task* dapat dihindarkan.

Saat ini RTOS yang disediakan oleh para pengembang perangkat lunak hampir bisa digunakan pada semua jenis mikrokontroler misalnya MCS51, AVR, PIC, ARM, ColdFire, x86, dan Blackfin. Jenis RTOS yang open source adalah CHIBI RTOS. Pada penelitian sebelumnya CHIBI RTOS telah berhasil diimplementasikan pada AVR sebagai mikrokontroler. CHIBI RTOS memfokuskan pengembangannya pada mikrokontroler jenis ARM [5].

Pada Tugas Akhir ini dilakukan implementasi CHIBI RTOS pada mikrokontroler setelah itu menganalisis RTOS dari aspek sifat dari *kernel* seperti *preemptive kernel*, *semaphore*, dan *clock tick* pada *kernel*. RTOS yang digunakan adalah CHIBI RTOS dan alat mikroprosesor yang digunakan adalah jenis ARM. Di negara Indonesia, implementasi CHIBI RTOS pada *embbeded system* Jenis ARM Cortex-M4 belum pernah diujikan dalam penelitian di lingkungan akademik Indonesia, sehingga perlu dilakukan dalam rangka mengawali kegiatan pengembangan selanjutnya. Sehingga tujuan dari makalah tugas akhir ini menjelaskan mengenai tahapan berikut yakni : (1) Mengimplementasikan CHIBI RTOS pada Mikrokontroler ARM jenis Cortex-M4; (2) Menganalisis kemampuan CHIBI RTOS pada Mikrokontroler ARM dengan melihat sifat dari *preemptive kernel* yaitu karakteristik 'responsif', *semaphore* yaitu karakteristik 'kendali penguna', dan *clock tick* pada *kernel* yaitu karakteristik 'deterministik'.

II. TINJAUAN PUSTAKA

2.1 Sistem Waktu-Nyata (*Real Time System*)

Real Time System atau Sistem Waktu-Nyata adalah sistem yang harus menghasilkan respon yang tepat dalam batas waktu yang telah ditentukan. Jika respon komputer melewati batas waktu tersebut, akan terjadi penurunan kemampuan atau kegagalan sistem[3].

2.2 Sistem Operasi Waktu-Nyata

Sistem operasi waktu-nyata (*Real-Time Operating Sistem*) merupakan perangkat lunak sistem yang berseluler mengatur *resource* yang disediakan oleh perangkat keras dan menyediakan fasilitas pemrograman untuk digunakan oleh aplikasi. Sistem operasi waktu nyata memiliki karakteristik yang berbeda dengan sistem operasi biasa, sehingga tidak semua sistem operasi bisa disebut sebagai sebuah sistem operasi waktu-nyata[3].

2.3 Sistem Operasi Waktu-Nyata CHIBI

CHIBI RTOS yang dikenal dengan logo seperti pada Gambar 2.2 merupakan salah satu dari sekian banyaknya sistem operasi waktu-nyata yang ada pada saat ini. CHIBI RTOS merupakan sistem operasi waktu-nyata yang menggunakan bahasa pemograman C dan C++. Untuk saat ini ChibiOS/RT sudah dapat di-*porting* ke dalam beberapa arsitektur mikrokontroler. Selain itu juga bisa digunakan sebagai simulator pada Windows x86[4].

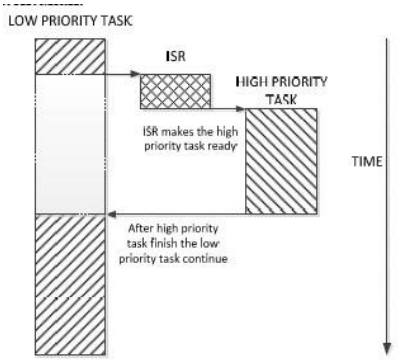
CHIBI RTOS adalah perangkat lunak gratis dengan lisensi GNU *general public license* 3

atau GPL3. Perangkat lunak ini bisa diperbanyak atau disebarluaskan kepada umum tetapi tidak boleh mengklaim hak cipta dari CHIBI RTOS dan tidak disarankan mengubah konfigurasi dan fungsi yang sudah ada (kecuali konfigurasi pada mikrokontroler yang digunakan).

Penggunaan CHIBI RTOS diperlukan mikrokontroler yang memenuhi spesifikasi minimum sebagai berikut : (a) Arsitektur minimum CPU dengan 8-bits; (b) Mendukung untuk bahasan standar C89 dan C99; (c) Memiliki RAM minimal 2KB; (d) Memiliki memori untuk program sebesar 16KB[4].

2.4 Premmptive Kernel

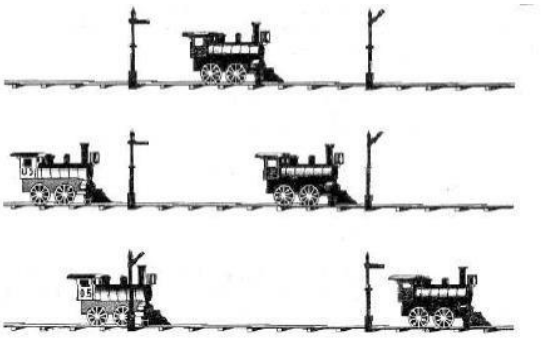
Kernel *preemptive* mengizinkan *preemption* pada sebuah proses yang berjalan di mode kernel. Implementasi dari kernel *preemptive* sangatlah sulit dan banyak aplikasi (*spreadsheets*, *word processors*, dan *web browsers*). *Kernel preemptive* tidak memerlukan *response time* yang cepat, namun untuk memenuhi persyaratan waktu pada sistem waktu nyata (terutama pada sistem waktu nyata keras) *kernel preemptive* menjadi sangat penting. Skema prinsip kerja *preemptive kernel* dapat lihat pada Gambar 2.1 berikut.



Gambar 2.1 Skema prinsip kerja *preemptive kernel* [2]

2.5 Semaphore

Sebagai mekanisme sinkronisasi, *semaphore* akan menjaga *task* yang berjalan secara konkuren untuk saling mengganggu ketika sedang mengakses sumber daya yang sama dan memberikan peringatan ketika terjadi suatu *error*. Jadi dalam sistem operasi waktu-nyata adalah *task* yang ingin mengakses *shared* data atau *shared hardware*. Visualisasi dari cara kerja *semaphore* dapat dilihat pada Gambar 2.2 berikut.

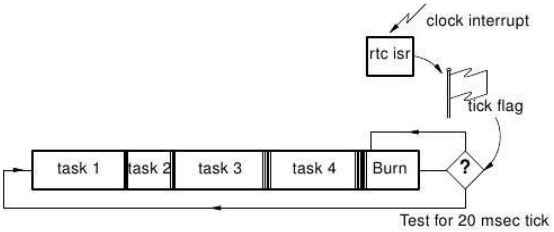


Gambar 2.2 Visualisasi dari cara kerja *semaphore* [5]

Jadi jika dalam RTOS maka lokomotif tersebut adalah task yang ingin mengakses *shared data* atau *shared hardwared*[5].

2.6 Clock Tick

Clock tick merupakan interupsi spesial yang muncul secara periodik. *Clock tick* adalah kunci utama dari sistem yang berfungsi sebagai dasar untuk menentukan *timer* pada sistem *real time* dengan sistem operasi waktu-nyata. Waktu untuk tiap munculnya *clock tick* bisa ditentukan oleh pada saat merancang sistem operasi waktu- nyata. Semakin cepat *clock tick*, semakin besar beban yang ditanggung oleh CPU. Semua *kernel* mampu untuk menunda *task* sesuai dengan nilai *clock tick*. Skema prinsip kerja *clock tick* dapat di lihat pada Gambar 2.3 berikut.



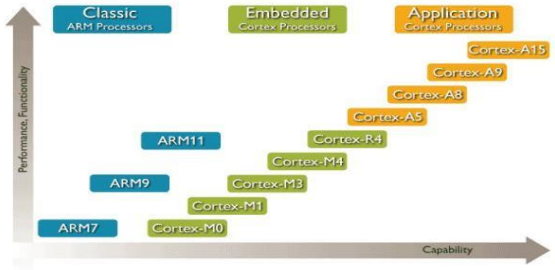
Gambar 2.3 Kehadiran *clock tick* [6]

2.7 Mikrokontroler ARM Cortex-M4

Mikrokontroler dapat dipandang sebagai sistem yang memiliki prosesor, memori, dan port I/O. Aplikasi pada Mikrokontroler sendiri seperti memantau *environment* untuk mendapat *input*, menghasilkan respons sebagai *output*, respons sendiri dapat ditunda untuk *timer*/*counter*, respons yang harus diprioritaskan untuk interupsi, *software* untuk mengontrol proses bagian memori, dan data sementara yaitu SRAM[7].

ARM (Gambar 2.4), adalah prosesor dengan arsitektur set instruksi 32 bit RISC

(*Reduced Instruction Set Computer*) yang dikembangkan oleh ARM Holding. ARM merupakan singkatan dari *Advanced* RISC *Machine*. ARM sendiri terdiri dari 3 karakter sebagaimana pada Gambar 2.4 yaitu ARM Klasik, ARM Embedded dan ARM Application. ARM yang dikembangkan secara khusus untuk keperluan *realtime* adalah ARM seri CortexM dan CortexR. Contoh penerapan CortexM dan CorteR adalah seperti mesin hidrolik, mesin otomotif, sensor penjadwalan, dan kontroler elektronik[7].



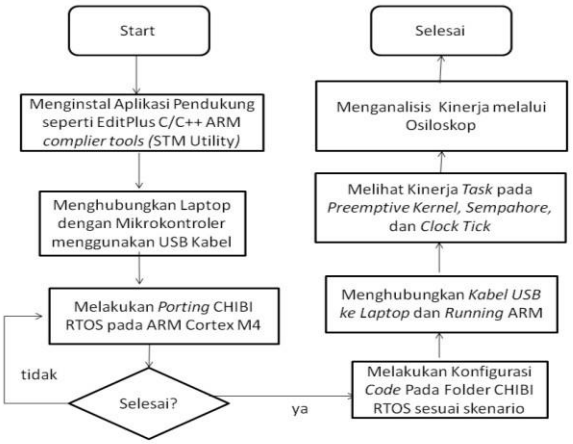
Gambar 2.4 Keluarga ARM mikroprosesor [12]

III. PERANCANGAN SISTEM

Prancangan sistem pada makalah ini terdiri dari 4 perancangan, yakni (1) Diagram alir tahapan rancangan sistem; (2) Perancangan perangkat keras; (3) Perancangan perangkat lunak; (4) Perancangan integrasi perangkat keras dan perangkat lunak.

3.1 Diagram Alir Tahapan Rancangan Sistem

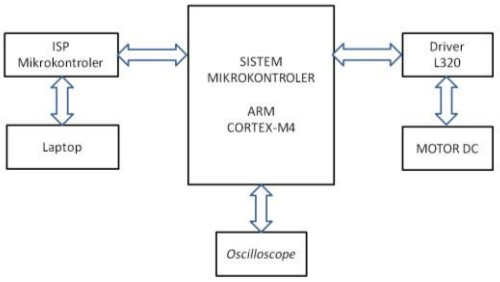
Pada tahap awal *file* CHIBI RTOS yang sudah di-*download* dipersiapkan, lalu dilakukan instalasi *software* yang mendukung dalam *code* bahasa pemrograman C/C++ yaitu EditPlus, dan STM32 ST-Link Utility yang mendukung untuk memasukkan *code* program ke dalam mikrokontroler ARM. Selanjutnya dilakukan meng-*code* sebuah program pengontrolan Motor DC dan LED pada File CHIBI RTOS. Setelah *code* selesai dibuat, selanjutnya dilakukan make file melalui *command prompt* (cmd). Setelah selesai melakukan make file, aplikasi STM32 *Link-Utility* dinyalakan serta Laptop dan mikrokontroler dihubungkan dengan kabel USB dalam hal ini ISP (*In System Programming*). Selanjutnya, *file* CHIBI RTOS yang telah dibuat ditanamkan ke dalam mikrokontroler ARM. *Push button* yang terdapat pada mikrokontroler ARM berguna untuk me-*reset* status mikrokontroler ARM untuk memulai Motor DC dan LED yang nyala. Proses jalannya task tersebut diatur dengan waktu tertentu seperti 4 *task* yang dijalankan dengan jenisnya masing-masing (*Absolute priority*, *High Priority*, *Normal priority*, dan *Low Priorty*). Pada hasil tersebut ditampilkan kinerja *Task* tersebut dalam waktu yang sudah dijadwalkan melalui *oscilloscope*. Tahap terakhir adalah menganalisis aspek *reliability* dan p*ortability* dari CHIBI RTOS pada ARM, dan keamanan (*safety*/*secure*) ketika menginterupsi sebuah *task*. Secara umum diagram alir tahapan rancangan sistem dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram alir rancangan sistem

3.2 Perancangan Perangkat Keras

Perangkat keras: sebagai perangkat yang digunakan dari sistem *real time* yang dirancang. Dalam perancangan perangkat keras, digunakan ARM jenis CortexR yang mendukung pemrosesan *real time*. Perangkat ini juga dilengkapi ISP (*In System Programming*) yang menghubungkan antara Laptop dengan *Embeded Systems* (Sistem Mikrokontroler). Modul sistem ARM jenis CortexR sendiri terdapat LED dan *push button*. Secara umum perancangan perangkat keras sistem dapat dilihat pada Gambar 3.2.

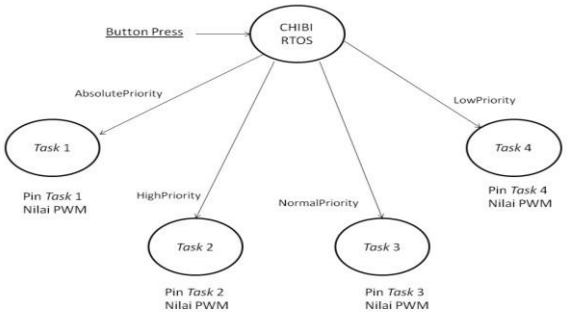


Gambar 3.2 Diagram sistem perangkat keras

3.3 Perancangan Perangkat Lunak

Sistem yang dibuat hanya berupa pembuktian dan penjelasan dari beberapa fungsi yang terdapat pada ChibiOS jika menggunakan

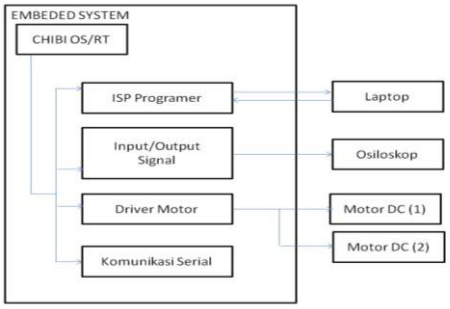
jenis mikrokontroler ARM. Sistem ini terdapat empat buah *task* dengan prioritas berbeda. Saat sistem berjalan, prirotas keempat *task* dapat berubah prioritas, dari perubahan tersebut dapat dilihat respons sistem terhadap tingkat prioritas. Secara umum perancangan perangkat lunak sistem dapat dilihat pada Gambar 3.3.



Gambar 3.3 Diagram perangkat lunak dari system

3.4 Perancangan integrasi perangkat keras dan perangkat lunak

Berdasarkan metode yang telah diungkapkan di atas, maka alur pembuatan sistem antara perangkat keras dan perangkat lunak dapat dilihat pada Gambar 3.4 berikut :



Gambar 3.5 Bagan Alur Jalannya Sistem

Embeded System di sini adalah mikrokontroler ARM Cortex-M4 dimana CHIBI RTOS adalah sistem operasi yang akan ditanam. Laptop melakukan komunikasi dengan Mikrokontroler ARM melalui kabel USB atau ISP Programmer yang berhubungan dengan Mikrokontroler. *Code* program ditulis di Laptop lalu dihubungkan kedalam ARM Cortex-M4. Disamping itu untuk menjalankan Motor DC, dibutuhkan sebuah *driver* motor untuk suplai tenaga ke Motor DC. Untuk menganalisis *Preemptive Kernel*, *Semaphore*, dan *Clock Tick* diamati melalui Osiloskop.Komunikasi Serial sendiri yang menghubungkan antara PIN ARM dengan *driver* motor.

3.5 Skenario Pengujian Preemptive kernel

Terdapat tiga buah skenario dengan tujuan dari ketiga skenario ini adalah melihat dan menganalisis sifat dari *kernel* yang digunakan yaitu *preemptive kernel.* Osiloskop digunakan untuk melihat *task* yang akan berlangsung. *Task* dengan prioritas tertinggi akan menunda *task* dengan prioritas yang lebih rendah. Secara umum rencana pengujian *preemptive kernel* dapat dilihat pada Tabel 3.1, Tabel 3.2 dan Tabel 3.3.

Tabel 3.1 *Timeline* rencana empat *task* dijalankan bersamaan

Task	Prioritas	Waktu	Waktu Tunda
1. ‘LED 1’	NORMAL	400ms	2000ms
2. ‘MOTOR DC 1’	NORMAL+1	400ms	2000ms
3. ‘MOTOR DC 2’	HIGH	200ms	1000ms
4. ‘LED 2’	ABSOLUTE	100ms	500ms

Tabel 3.2 *Timeline* rencana empat *task* dijalankan bersamaan

Task	Prioritas	Waktu	Waktu Tunda
1. ‘LED 1’	NORMAL	400ms	0ms
2. ‘MOTOR DC 1’	NORMAL+1	400ms	0ms
3. ‘MOTOR DC 2’	HIGH	200ms	0ms
4. ‘LED 2’	ABSOLUTE	100ms	0ms

Tabel 3.3 *Timeline* rencana empat *task* dijalankan bersamaan

Task	Prioritas	Waktu	Waktu Tunda
1. ‘LED 1’	NORMAL	400ms	0ms
2. ‘MOTOR DC 1’	NORMAL+1	400ms	0ms
3. ‘MOTOR DC 2’	HIGH	200ms	0ms
4. ‘LED 2’	ABSOLUTE	100ms	0ms

3.6 Skenario Pengujian Semaphore

Terdapat dua buah skenario dalam pengujian Semaphore. Tujuan dari kedua skenario ini adalah melihat dan menganalisis sifat dari *semaphore.* Pada skenario pertama osiloskop digunakan untuk melihat perbandingan 2 buah *task* yang akan berlangsung. Pada pengujian dicoba menjalankan *task* 1 yang prioritas *high* dan task 4 yang prioritas *normal* agar hasilnya bisa teratur saat memulai pengerjaan *task*. Pada pengujian *semaphore* di penelitian ini dicoba memasukkan fungsi API '*chBsemWait()'* pada *task* 1 yang memiliki prioritas *high* dan fungsi API '*chBSemSignal'* pada akhir pengerjaan *task* 4 yang memiliki proritas *normal*. Pada *task* 1 waktu pengerjaannya 500 milidetik dengan waktu tunda 1000 milidetik, sedangkan pada *task* 4 waktu pengerjaan 500 milidetik dengan waktu tunda 1000 milidetik. Pada skenario kedua kedua *task* disamakan prioritasnya menjadi *normal*. Secara umum pengujian *semaphore* dapat dilihat pada Tabel 3.4 untuk skenario pertama dan Tabel 3.5 untuk skenario kedua.

Tabel 3.4 *Timeline* rencana dua *task* dijalankan bersamaan

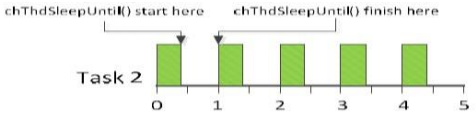
Task	Prioritas	Waktu	Waktu Tunda
1. 'LED 1'	HIGH	500ms	1000ms
4. 'LED 2'	NORMAL	500ms	1000ms

Tabel 3.5 *Timeline* rencana dua *task* dijalankan bersamaan

Task	Prioritas	Waktu	Waktu Tunda
1. 'LED 1'	NORMAL	500ms	1000ms
4. 'LED 2'	NORMAL	500ms	1000ms

3.7 Skenario Pengujian Clock Tick

Tujuan dari pengujian ini adalah melihat dan menganalisis sifat dari *clock tick. Clock tick* pada *scheduling* untuk tiap *task* menggunakan fungsi API '*chTimeNow*()' yang memulai *timer* 0. Pada tahap ini disimpan *tick* yang terjadi sejak awal sistem berjalan sampai sistem berakhir. *Tick* dari sistem disimpan ke dalam variabel *time*, yang dapat ditambahkan dengan periode yang diinginkan. Kemudian fungsi API '*chThdSleepUntil*()' membuat *task* tertunda selama periode yang dinginkan.



Gambar 3.8 Contoh *timeline* fungsi API *chThdSleepUntil*() [2].

IV. IMPLEMENTASI DAN ANALISIS

4.1 Implementasi RTOS pada Mikrokontroler

Pada tahap implementasi terlebih dahulu dilakukan konfigurasi CHIBI RTOS agar sesuai dengan arsitektur Mikrokontroler ARM dengan cara *Porting*. Setelah itu dilakukan penulisan kode program yang bertujuan untuk menjalankan LED dan Motor DC nya, kode program ditulis melalui aplikasi EditPlus. Setelah kode program ditulis, selanjutnya dilakukan *make file* sehingga menghasilkan ch.bin. Selanjutnya Laptop dihubungkan dengan Mikrokontroler ARM melalui kabel USB ditanam *file* ch.bin ke dalam Sistem Mikrokontroler. *File* ch.bin ditanam dalam Mikrokontroler melalui aplikasi STM32 ST-LINK Utility.

Pada *Porting mode* PWM diaktivkan terlebih dahulu pada konfigurasi dalam file halconf.h yang terdapat pada *folder* CHIBI RTOS bagian ST32F4 *Discovery*. *Mode* PWM yang awalnya FALSE diubah menjadi TRUE.

```
#if !defined(HAL_USE_PWM) ||
defined(__DOXYGEN__)
#define HAL_USE_PWM          TRUE
#endif
```

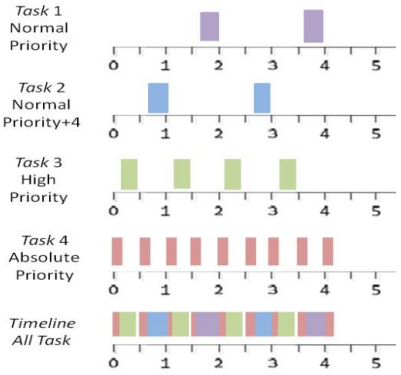
Pada tahap ini juga dilakukan konfigurasi kode untuk Motor DC, Lampu LED Mikrokontroler ARM, kode *preemptive kernel* untuk mengatur prioritas, dan kode *semaphore*.

4.2 Hasil Pengujian

Pengujian sistem yang dibuat adalah untuk mengamati *preemtive kernel*, *semaphore* dan *clock tick*. Pengujian tersebut juga adalah pengujian sistem RTOS yang telah ditanam pada mikrokontroler ARM CORTEX M4 dan melakukan pengontrolan pada motor DC dan lampu LED.

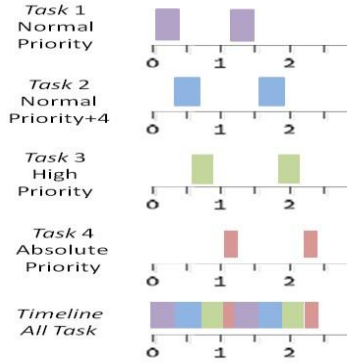
4.2.1 Pengujian *Preemptive Kernel*

Karakteristik dan data uji skenario sesuai dengan Tabel 3.1, Tabel 3.2 dan Tabel 3.3. Skenario pertama dan kedua berjalan, sedangkan pada skenario ketigat tidak dapat berjalan. Berikut untuk melihat *preemptive kernel* nya 2 buah *task* diamati bersama dalam sebuah layar osiloskop 2 chanel dan dikonversi pada Gambar 4.1 dan pada Gambar 4.2. Sedangkan pada skenario 3 yang tidak berjalan dapat dilihat pada osiloskop Gambar 4.3



Gambar 4.1 Bagan skenario pertama untuk *timeline preemptive kernel*

Pada percobaan skenario 1 untuk pengerjaan *task* 1, *task* 2, *task* 3 dan *task* 4 terlihat bahwa dari *word execution time* tetap dijalankan bersamaan. Pada percobaan ini terlihat *Preemptive* Kernel bekerja dengan baik pada *task* 1, *task* 2, *task* 3 dan *task* 4 sesuai dengan perintah waktu pengerjaan dan waktu tunda yang ada pada *code*. *Task* 4 yang '*absolute priority'* lebih dulu dikerjakan, lalu *task* 3 '*high priority'* dikerjakan, selanjutnya '*normal priority*+4' dan terakhir *task* 4 yang '*normal priority'*. Adanya sela muncul sesekali dikarenakan adanya waktu tunda. Dapat disimpulkan bahwa *preemptive kernel* dapat bekerja dengan baik dengan urutan skala prioritasnya



Gambar 4.2 Bagan skenario kedua untuk *timeline preemptive kernel*

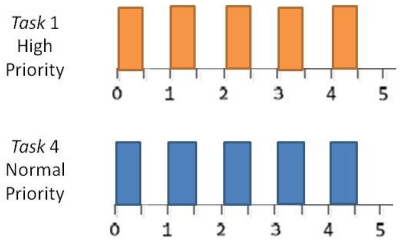
Pada percobaan skenario 2 ini terlihat *Preemptive* Kernel bekerja dengan baik pada *task* 1, *task* 2, *task* 3 dan *task* 4 sesuai dengan perintah waktu pengerjaan dan tidak terdapat sela waktu perpindahan. Dimana urutan pengerjaannya dimulai dari *task* 1, *task* 2, *task* 3 dan terakhir *task* 4. Disini tidak terdapat sela diantara pergantian *task* karena waktu tunda nya ditiadakan. Dapat disimpulkan bahwa kehadiran waktu tunda akan mengatur urutan pengerjaan pada sebuah *task* dengan prioritas. *Preemptive kernel* dapat berjalan apabila terdapat prioritas yang berbeda.

Gambar .4.3 Bagan timeline skenario tiga

Pada percobaan skenario 3 ini terlihat *Preemptive* Kernel tidak dapat bekerja pada *task* 1, *task* 2, *task* 3 dan *task* 4 sesuai dengan perintah waktu pengerjaan, dikarenakan 4 buah task bersamaan diberikan prioritas yang sama, yaitu 'normal priority', sehingga hanya 1 task yang berjalan sedangkan task lainnya tidak dapat berjalan.

4.2.2 Pengujian *Semaphore*

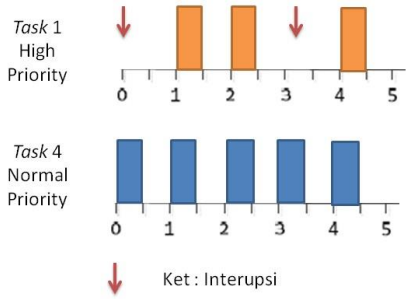
Karakteristik dan data uji sesuai dengan Tabel 3.4 dan Tabel 3.5. Pada pengujian mula-mula dijalankan *task* 1 dan *task* 4 bersamaan dengan fungsi *semaphore* belum diaktivkan sehingga hasil pengujian untuk *timeline* saat *task* 1 dan *task* 4 dijalankan secara bersamaan dapat dilihat pada Osiloskop yang dikonversikan pada Gambar 4.4.



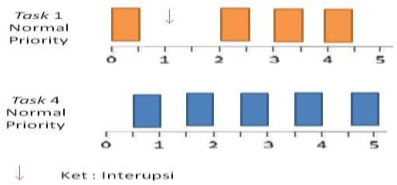
Gambar 4.4 *Timeline* untuk *task* 1 dan *task* 4 tanpa semaphore

Pada *task* 1 (warna kuning) memiliki prioritas yang lebih tinggi daripada *task* 4 berjalan bersamaan tanpa ada nya *code semaphore*. Selanjutnya fungsi *semaphore* digunakan dengan

cara dilakukan interupsi saat *task* 1 dan *task* 4 sedang berjalan bersamaan, maka hasil yang didapatkan terlihat pada Osiloskop yang dikonversikan pada Gambar 4.5 dan 4.6.



Gambar 4.5 *timeline* skenario pertama untuk *task* 1 dan *task* 4 jika menggunakan *semaphore*



Gambar 4.6 *timeline* skenario kedua untuk *task* 1 dan *task* 4 jika menggunakan *semaphore*

Pada *task* 1 (warna kuning) memiliki prioritas yang lebih tinggi daripada *task* 4. Ketika *Task* 1 dan *task* 4 sedang berjalan ditekan *push button* (interupsi) pada Mikrokontroler ARM, sehingga dapat dilihat bahwa *task* 1 berhenti sesaat menunggu sebuah proses *task* 4 berjalan dulu setelah baru *task* 1 berjalan kembali. Pada *code task* 1 terdapat fungsi API chBsemWait() dan *semaphore* sedang berada pada *task* 4 maka *task* 1 sebelum dikerjakan akan menunggu hingga *task* 4 mencapai fungsi API chBSemSignal(). Jika *task* 4 sudah melepas signal maka *task* 1 dapat berjalan dan pengerjaan *task* 1 dan *task* 4 mengikuti skala prioritasnya. Hasil pengujian mengenai *semaphore* sesuai dengan teori.

Hasil pengujian pada skenario pertama mengenai *semaphore* berbeda dengan pada skenario kedua karena pada skenario pertama pengerjaan dua buah *threads* tiap *task* dengan prioritas sama, sedangkan pengujian skenario kedua mengenai *semaphore* dengan prioritas sama. Dapat diketahui bahwa tugas *semaphore* adalah untuk penggunaan *resource* dalam *task* yang memiliki prioritas yang sama.

4.2.3 Pengujian *Clock Tick*

C*lock tick* terdapat automatis pada CHIBI RTOS cara penggunaann adalah menulis *code* chSysLockFromIsr(); dan chSysUnlockFromIsr(); pada static void PWMDriver. Nilai *tick* yang diperoleh pada ARM adalah saat PWM HIGH berpindah ke PWM NORMAL. Nilai satu *tick* yang berjalan di isi dengan 10kHz ke dalam PWMConfig nya dan Priodenya pada *prescaler* 128. Nilai Frekuensi yang ada pada grafik bisa dilihat pada Gambar 4.4 berikut.

Gambar 4.4 Perbedaan perpindahan *tick* (skala 10milidetik)

V. PENUTUP

5.1 Kesimpulan

Berdasarkan penelitian diatas, dapat disimpulkan bahwa :

1. CHIBI RTOS dapat diimplementasi dengan baik pada Mikrokontroler ARM jenis Cortex-M4 tipe STM32F4-Discovery.
2. Kemampuan yang dihasilkan oleh CHIBI RTOS pada ARM sesuai dengan keinginan. Pada pengamatan melalui Osiloskop terlihat *task* berjalan dengan urutan yang diminta dalam *code* dan tidak terdapat nilai *eror* baik pada *preemptive kernel* dan *semaphore*. Pada *clock tick* nilai yang didapatkan senilai 21 milidetik.
3. Ada tiga karakteristik sistem waktu nyata telah berhasil di implementasikan pada CHIBI RTOS dengan ARM yaitu : (1) Deterministik, waktu yang dieksekusi *clock tick* nya diketahui bernilai 21 milidetik; (2) Responsif, *preemptive kernel* melakukan eksekusi dengan waktu yang sama sesuai skala prioritasnya; (3) Kendali Penguna pada RTOS yaitu pada *semaphore* berhasil dilaksanakan apabila skala prioritasnya sama.

5.2 Saran

Dari penelitian yang dilakukan, maka diperoleh analisis yang akhirnya menghasilkan beberapa saran yang diharapkan dapat memberikan manfaat bagi pihak yang mungkin akan melakukan penelitian lebih lanjut dengan topik atau uji analisis yang sama, yaitu:

1. Pengujian pada penelitian ini hanya meninjau dari *Preemptive Kernel*, *Semaphore* dan *Clock Tick* pada Kernel sehingga untuk penelitian berikutnya dapat dilakukan berbagai uji coba dengan fitur lain seperti *Duty Cycle* pada PWM atau PAL nya.
2. Uji coba CHIBI RTOS dengan ARM Cortex-M4 tidak hanya dilakukan dengan Motor DC dan LED, namun juga dapat diujikan pada berbagai fitur perangkat yang lebih kompleks misalnya berbagai fitur sensor atau indera.
3. Untuk Penelitian selanjutnya sebaiknya membandingkan CHIBI RTOS dengan RTOS lainnya misalnya CooCoox, FreeRTOS, QNX Neutrino dan lainnya pada Mikrokontroler ARM yang sama.
4. Penggunaan CHIBI RTOS dengan ARM Cortex-M4 diharapkan langsung dapat diaplikasikan pada berbagai model instrumen. Contohnya : Sensor kepadatan *traffic light*

DAFTAR PUSTAKA

[1]Leksono, A.B., Setiawan, I., Setiyono,. B. 2011. *Penerapal Real Time Operating Systems Pada Mikrokontroler AVR (Studi Kasus CHIBIOS/RT)*. Makalah Seminar Tugas Akhir, Universitas Diponegoro, Semarang.

[2]Pamungkas, H., Setiawan, I., Setiyono,. B. 2011. *Desain dan Impelementasi Perangkat Lunak pada Sistem Mikrokontroler AVR berbasis CHIBIOS/RT (Studi Kasus Pengontrolan Motor DC).* Makalah Seminar Tugas Akhir, Universitas Diponegoro, Semarang.

[3]Laplante, P A. 2008. *RealTime Systems Design and Analysis*. Wiley-IEEE Press.

[4] -------, ChibiOS/RT Forum, http://forum.chibios.org/

[5]Kurniawan, FX R. 2011. *Multitasking pada Mikrokontroler ATmega16 menggunakan Real Time Operating System (RTOS) Jenis Cooperative*. Indonesia, Semarang : Universitas Diponegoro.

[6]Migliavacca, M. 2011. *Informatics for industrial applications Lecture 10 - ChibiOS/RT*. University of Milano-Bigcocca.

[7] -------, ARM *Product and Tools*, http://www.arm.com

[8]Simon, D E., 2005. *An Embedded Software Primer*, Pearson Education, Inc., India.

[9] -------, ChibiOS/RT Forum, http://forum.chibios.org/, Maret 2016

[10] ------, ChibiOS/RT Documentation and Guides, http://wwwchibios.org/dokuwiki/doku.php?id=chibios:documents, Maret 2016

[11] -------, ARM *Product and Tools*, http://www.arm.com, Maret 2016

[12]-------, RTOS ThreadX, http://rtos.com/downloads/articles_and_whitepapers-1/, Februari 2016

[13] ------, ARM Complier, http://www2.keil.com/mdk5/compiler/5/, Maret 2016

[14] ------, EditPlus - textEditor for Windows, https://www.editplus.com/, Maret 2016

[15] ------, Priority Task, http://www.chibios.com/forum/viewtopic.php?t=2953, Maret 2016

[16] ------, 3.0.2 Update STM32F4, http://www.chibios.com/forum/viewtopic.php?t=2707, Maret 2016

[17] 'Pengertian dan Kegunaan Osiloskop', http://teknikelektronika.com/pengertian-osiloskop-spesifikasi-penentu-kinerjanya/, Juni 2016